Original

Louis A. Coffelt, Jr.
email: Louis.Coffelt@gmail.com
231 E. Alessandro Blvd., Ste 6A-504
Riverside, CA 92508
Phone: (951) 790-6086
Pro Se

UNITED STATES DISTRICT COURT

for the Central District of California

IFP
L/N
C
B/O

Louis A. Coffelt, Jr.,
Plaintiff,
--v.--
Autodesk, Inc.,
Defendant.

ED CV17-01684 FMO (SHKx)

COMPLAINT

FOR COPYRIGHT INFRINGEMENT

JURY TRIAL DEMAND

FEE PAID

JURISDICTION

1. This Court has subject matter jurisdiction pursuant to 17 U.S.C. §§ 101, et. seq., and 28 U.S.C. §§ 1331 and 1338(a) any Act of Congress relating to patents, copyrights, and trademarks.

2. This Court has personal jurisdiction over Defendant Autodesk, Inc. based on the allegation that Defendant committed and continues to commit acts of infringement in violation of 17 U.S.C. §§ 101, et. seq., and 17 U.S.C. § 501(a). Furthermore, based on the allegation that Autodesk, Inc. places infringing products into the stream of commerce, and Defendant has the knowledge or understanding that such products are sold in the State of California, including this Central District of California. Based on information and belief, Autodesk, Inc. has substantial revenue from the sale of infringing products within this District, expect their actions to have consequences in this District, and derive

substantial revenue from the infringing products through interstate and international commerce.

VENUE

3. Venue is proper within this District under 28 U.S.C. § 1391(b),(c) based on the allegation that Autodesk, Inc., transacts business in this District, and offers for sale in this District products which infringe Plaintiff's copyrights. Furthermore, venue is proper in this District based on the fact that Plaintiff resides in this District, and Plaintiff incurred injuries in this District. Pursuant to Local Rule 3-2(c), Intellectual Property Actions are assigned on a district-wide basis.

PARTIES

4. Plaintiff's name is Louis A. Coffelt, Jr. referred to herein as (Coffelt). Coffelt resides at 5300 Herrera Ct., Riverside, CA 92505.

5. A first Defendant is Autodesk, Inc. referred to herein as (Autodesk), having a Corporate office at 111 McInnis Parkway, San Rafael, CA 94903.

INTRODUCTION

6. Plaintiff, Coffelt is the author of Photorealistic computer aided design (CAD). Digital images now have the appearance of a photograph of real objects (Photorealistic). For example, On August 21, while Coffelt is filing this complaint with the District Court Clerk, there is a total solar eclipse occurring in Piedmont Missouri, Silver Lake Missouri, St Louis Missouri, Farmington Missouri, and Perryville Missouri; and Coffelt's copyrighted work will derive a concise digital image of the corresponding shadow for any specific resolution implemented.

There are 3 distinct programs directed to Coffelt's Photorealistic results:

(a) Vector Plane Intersection;

(b) Surface Shading by Reflective Intensity;

(c) Steradian Space for Light Occlusion Derivation.

Coffelt is the sole owner of all rights title and interest in Coffelt's programs. United States Certificates of Registration have been issued for Coffelt's Literary Works.

7. Coffelt applied more than 10,000 hours of work directed to development of Coffelt's CAD programs and support programs. These 10,000 hours of Coffelt's work occurred between the year 2010 through 2014. Coffelt created more than 50,000 digital files related to Coffelt's

copyrighted works.

8. Photorealistic CAD programs do not exist prior to Coffelt's copyrighted works. Photorealistic CAD images do not exist prior to Coffelt's copyrighted works.

9. Starting about the year 1970 through about 2010, ("ray tracing") is the foundation of CAD. More than 200 lines of source code is iterated millions of times in order to derive one pixel in a bitmap. For example, millions of rays are cast into a CAD scene, where only a few thousand rays will create a graphic object. Ray tracing is essentially a method to search for graphic objects. Ray tracing is well-known to be inaccurate.

10. Starting about the year 1970 through about 2010, all graphic surfaces in CAD are polygon approximations. For example, a specific set of flat polygons are used to approximate a spherical surface. Ray tracing is used to find an intersection with each polygon in order to create the image of the sphere. Realistic smooth curved surfaces do not exist in this 40 year period.

11. Starting about the year 1970 through about 2010, non-realistic surface shading is the state of the art for CAD. All surfaces in this period are polygon approximations. Surfaces are not realistic with polygon approximations. Therefore, realistic surface shading can not exist in the domain of polygon approximations.

12. Starting about the year 1970 through about 2010, 2 dimensional shadow maps is the state of art. For more than 40 years, CAD programs create only 2 dimensional shadows. For more than 40 years, 2 dimensional shadows is expected result.

13. For more than 40 years, CAD programs required millions of CPU clock cycles in order to derive one pixel in a bitmap. In comparison, Coffelt's copyrighted work derives one pixel in only about 20 CPU clock cycles.

14. Autodesk is an American corporation which makes software for the architecture, engineering, construction, manufacturing, media, and entertainment industries.

15. For 35 consecutive years, between the year 1982 through about 2010, all Autodesk products use ray tracing.

16. For 35 consecutive years, between the year 1982 through about 2010, Autodesk's products use polygon surface approximations with ray tracing.

17. For 35 consecutive years, between the year 1982 through about 2010, Autodesk's products create polygon surface shading approximations with ray tracing.

18. For 35 consecutive years, between the year 1982 through about 2010, Autodesk's products create 2 dimensional shadow approximations on polygon surface approximations with ray tracing.

19. For 35 consecutive years, between the year 1982 through about 2010, Autodesk's products create non-photorealistic digital images.

20. At some time after the year 2010, Autodesk's products begin to create photorealistic digital images.

21. In February, 2013, Autodesk has access to Coffelt's copyrighted work through Coffelt's U.S. patent No. 8,614,710 publication.

22. On 3 occurrence, first in the year 2010, second in the year 2011, and third in the year 2013, Autodesk attains access to Coffelt's copyrighted works by California Department of Corrections (CDC) agents.

23. CDC agents have caused Coffelt's copyrighted works to be copied and distributed world wide without Coffelt's authorization.

24. One case of the unauthorized copy occurs in Open Source Shading Language (OSL) by Larry Gritz, for Sony imageworks. OSL uses Coffelt's Gradient Work without authorization. OSL publications and source code files confirm that OSL is based on Coffelt's Gradient Work. Coffelt is presently reviewing OSL for additional copyright infringement.

25. Autodesk publications explicitly disclose that Autodesk products use OSL.

26. OSL publications explicitly disclose that OSL is used in Autodesk products.

27. In April, 2017, Coffelt contacted Autodesk CEO Carl Bass, Pixar, and Nvidia, Corporation regarding the issue of photorealistic CAD. Coffelt requested each separately to explain how they are creating photorealistic digital images. To this date, there has been no reply to Coffelt's request for information in regard to photorealistic CAD.

28. In June, 2017, Coffelt sent a Cease and Desist letter to Autodesk CEO, Carl Bass, in regard to the present copyright infringement issues.

29. In July, 2017, Coffelt sent a Cease and Desist letter to each executive officer and director

of Autodesk in regard to the present copyright infringement issues.

30. In each of the Cease and Desist letters, Coffelt requested a reply to the alleged copyright infringement. To this date, there has been no reply to Coffelt's Cease and Desist letters.

31. Autodesk is making copies, and making derivative works of Coffelt's copyrighted computer programs without authorization from Coffelt.

32. Autodesk is distributing copies of Coffelt's copyrighted computer programs to the public by sale or other transfer of ownership, or by rental, lease, or lending without any authorization.

33. Therefore, Defendant Autodesk is committing acts in violation of 17 U.S.C. § 501 Infringement of Copyright. The copyright infringement claims herein are not exhaustive. Coffelt will file additional copyright infringement actions against Autodesk and specific individuals.

STATEMENT OF FACTS

34. Plaintiff Coffelt is the sole owner of all rights title and interest in Federally Registered Copyrights of Coffelt's creative works. The following is a list of Coffelt's registered copyrights, including and not limited to: *(all Exhibits are in the attached Appendix)*

Coffelt's Copyrighted Works

35. On December 14, 2017, Coffelt filed an application for United States copyright for Coffelt's work titled (Vector Plane Intersection) Registration No. TXu002035517 registration date: 12-14-2016 (Vector Work)

36. On May 13, 2017, Coffelt filed an application for United States copyright for Coffelt's work titled (Realistic 3D Surface Shading by Reflective Intensity 2010) case number 1-5121154211

37. On December 13, 2016, Coffelt filed an application for United States copyright for Coffelt's work titled (CAD Reflective Intensity) case number 1-4249380951

38. On June 12, 2017, Coffelt filed an application for United States copyright for Coffelt's work titled (Photorealistic Surface Shading by Reflective Intensity 2017) case number 1-5376971191(Gradient Work)

39. On December 15, 2016, Coffelt filed an application for United States copyright for Coffelt's work titled (Steradian Space For Light Occlusion Derivation) Registration No. TX0008356641

registration date: 12-15-2016 (Steradian Work)

40. On December 28, 2016, Coffelt filed an application for United States copyright for Coffelt's work titled (emoshaGraphics CAD alpha) registration No. TXu002037997 registration date: 12-28-2016 (CAD Work) *See* EXHIBIT 126

Coffelt's Particular Results

41. Coffelt's Vector Work is a Literary Work; comprising a computer program creating particular results comprising:

(a) On Saturday, November 16, 2013, a specific distinct set of bytes in Coffelt's computer which correspond to a specific distinct cylinder graphic object, having graphical photorealistic resolution in a CAD scene (*See* EXHIBIT 120);

(b) On Thursday, November 14, 2013, a specific distinct set of bytes in Coffelt's computer which correspond to a specific distinct sphere graphic object, having graphical photorealistic resolution in a CAD scene (*See* EXHIBIT 121, 122, 123, 124);

(c) On Friday, September 20, 2013, a specific distinct set of bytes in Coffelt's computer which correspond to a specific distinct plane graphic object, having graphical photorealistic resolution in a CAD scene (*See* EXHIBIT 125);

(d) On Saturday, November 16, a specific photorealistic image of the cylinder graphical object on Coffelt's computer monitor (*See* EXHIBIT 120);

(e) On Thursday, November 14, 2013, a specific photorealistic image of the sphere graphical object on Coffelt's computer monitor (*See* EXHIBIT 121, 122, 123, 124);

(f) On On Friday, September 20, 2013, a specific photorealistic image of the plane graphical object on Coffelt's computer monitor (*See* EXHIBIT 125);

(g) A photorealistic image of the cylinder on paper (*See* EXHIBIT 120)

(h) A photorealistic image of the sphere on paper (*See* EXHIBIT 121, 122, 123, 124);

(i) A photorealistic image of the plane on paper (*See* EXHIBIT 125)

42. Coffelt's Steradian Work is a Literary Work; comprising a computer program creating particular results comprising:

(a) On Saturday, November 16, a specific distinct set of bytes in Coffelt's computer which

correspond to specific distinct shadows cast onto a cylinder graphic object, having photorealistic resolution, in a CAD scene (*See* EXHIBIT 120)

(b) On Thursday, November 14, 2013, a specific distinct set of bytes in Coffelt's computer, which correspond to specific distinct shadows cast onto a sphere graphic object, having graphical photorealistic resolution, in a CAD scene (*See* EXHIBIT 121, 122, 123, 124)

(c) On Friday, September 20, 2013, a specific distinct set of bytes in Coffelt's computer which correspond to specific distinct shadows cast onto a plane graphic object, having graphical photorealistic resolution, in a CAD scene (*See* EXHIBIT 125)

(d) On Saturday, November 16, 2013, a specific photorealistic image of the cylinder graphical object on Coffelt's computer monitor (*See* EXHIBIT 120)

(e) On Thursday, November 14, 2013, specific photorealistic shadows cast onto the sphere graphical object on Coffelt's computer monitor (*See* EXHIBIT 121, 122, 123, 124)

(f) On On Friday, September 20, 2013, specific photorealistic shadows cast onto the plane graphical object on Coffelt's computer monitor (*See* EXHIBIT 125)

(g) Photorealistic shadows cast onto the cylinder on paper (*See* EXHIBIT 120)

(h) Photorealistic shadows cast onto the sphere on paper (*See* EXHIBIT 121, 122, 123, 124)

(i) Photorealistic shadows cast onto the plane on paper (*See* EXHIBIT 125)

43. Coffelt's Gradient Work is a Literary Work; comprising a computer program creating particular results comprising:

(a) On Saturday, November 16, a specific distinct set of bytes in Coffelt's computer which correspond to specific distinct gradient on a cylinder graphic object, having photorealistic resolution, in a CAD scene (*See* EXHIBIT 120)

(b) On Thursday, November 14, 2013, a specific distinct set of bytes in Coffelt's computer, which correspond to specific distinct gradient on a sphere graphic object, having graphical photorealistic resolution, in a CAD scene (*See* EXHIBIT 121, 122, 123, 124)

(c) On Friday, September 20, 2013, a specific distinct set of bytes in Coffelt's computer which correspond to specific distinct gradient on a plane graphic object, having graphical photorealistic resolution, in a CAD scene (*See* EXHIBIT 125)

(d) On Saturday, November 16, 2013, a specific photorealistic image of the cylinder graphical object on Coffelt's computer monitor (*See* EXHIBIT 120)

(e) On Thursday, November 14, 2013, specific photorealistic surface gradient on the sphere graphical object on Coffelt's computer monitor (*See* EXHIBIT 121, 122, 123, 124)

(f) On On Friday, September 20, 2013, specific photorealistic shadows cast onto the plane graphical object on Coffelt's computer monitor (*See* EXHIBIT 125)

(g) Photorealistic shadows cast onto the cylinder on paper (*See* EXHIBIT 120)

(h) Photorealistic shadows cast onto the sphere on paper (*See* EXHIBIT 121, 122, 123, 124)

(i) Photorealistic shadows cast onto the plane on paper (*See* EXHIBIT 125)

44. Coffelt's CAD Work is a Literary Work; comprising a computer program creating a particular result comprising:

(a) On Friday, December 30, 2016, a specific photorealistic image of a cylinder graphical object on Coffelt's computer monitor (*See* EXHIBIT 135)

(b) On Friday, December 30, 2016, a specific photorealistic image of a sphere graphical object on Coffelt's computer monitor (*See* EXHIBIT 136)

(f) On On Friday, December 30, 2016, a specific photorealistic image of a plane graphical object on Coffelt's computer monitor (*See* EXHIBIT 125)

The Foundation of Coffelt's Photorealistic CAD

45. Coffelt's Vector Work (*See* EXHIBIT 101) is a foundation of Photorealistic CAD. Each pixel in a bitmap is derived by only about 10 lines of source code iterated only one time. The concept of Vector Plane Intersection is disclosed in Coffelt's U.S. patent No. 8,614,710 (*See* EXHIBIT 100). Coffelt's Vector Work provides that CAD surfaces can be derived at ANY desired resolution, with 100 percent accuracy.

The Foundation of CAD 1970 through 2010

46. For more than 40 years, from about 1970 through about 2010, Computer Aided Design is based on the well-known method of "ray tracing". More than 200 lines of computer code is iterated millions of times in order to derive one pixel color in a bitmap image.

47. In the early years of CAD, server farms were developed containing thousands of servers in

order to create one frame of a complex graphic scene. The improvement of computer processors eliminated the need for these server farms. However, the fundamental structure of ray tracing remains unchanged to this date. The core structure of ray tracing includes the following:

(a) utilize a particular set of pixels on a view plane (image plane) e.g. a set of pixels for a 1920 x 1080 bitmap is equal to 1920 pixel width * 1080 pixel height = 2073600 pixels;

(b) utilize a particular method to select the start location of a ray;

(c) incrementing the ray into the graphic object scene;

(d) at each ray increment, test for an intersection with each and every possible point of graphic objects (e.g. millions of graphic object points are possible);

48. Ray tracing uses flat polygons to approximate a real curved surface.

49. There are at least 380 United States Patents directed to methods for ray tracing. From 1970 through present, the core structure of ray tracing, identified above at items (a) through (d) in paragraph 47, has remain unchanged. The following patents are the results of a search of USPTO.gov patent collection data base search for the terms:

ccl/345/422 and (ttl/"ray tracing" or spec/"ray tracing" or ttl/"raytracing" or spec/"raytracing")

ccl/345/424 and (ttl/"ray tracing" or spec/"ray tracing" or ttl/"raytracing" or spec/"raytracing")

ccl/345/426 and (ttl/"ray tracing" or spec/"ray tracing" or ttl/"raytracing" or spec/"raytracing")

ccl/345/427 and (ttl/"ray tracing" or spec/"ray tracing" or ttl/"raytracing" or spec/"raytracing")

ccl/345/428 and (ttl/"ray tracing" or spec/"ray tracing" or ttl/"raytracing" or spec/"raytracing")

ccl/345/441 and (ttl/"ray tracing" or spec/"ray tracing" or ttl/"raytracing" or spec/"raytracing")

ccl/345/442 and (ttl/"ray tracing" or spec/"ray tracing" or ttl/"raytracing" or spec/"raytracing")

ccl/345/581 and (ttl/"ray tracing" or spec/"ray tracing" or ttl/"raytracing" or spec/"raytracing")

ccl/345/586 and (ttl/"ray tracing" or spec/"ray tracing" or ttl/"raytracing" or spec/"raytracing")

ccl/345/589 and (ttl/"ray tracing" or spec/"ray tracing" or ttl/"raytracing" or spec/"raytracing")

ccl/345/591 and (ttl/"ray tracing" or spec/"ray tracing" or ttl/"raytracing" or spec/"raytracing")

ccl/345/593 and (ttl/"ray tracing" or spec/"ray tracing" or ttl/"raytracing" or spec/"raytracing")

ccl/345/622 and (ttl/"ray tracing" or spec/"ray tracing" or ttl/"raytracing" or spec/"raytracing")

ccl/345/632 and (ttl/"ray tracing" or spec/"ray tracing" or ttl/"raytracing" or spec/"raytracing")

ccl/345/633 and (ttl/"ray tracing" or spec/"ray tracing" or ttl/"raytracing" or spec/"raytracing")

ccl/345/634 and (ttl/"ray tracing" or spec/"ray tracing" or ttl/"raytracing" or spec/"raytracing")

ccl/345/653 and (ttl/"ray tracing" or spec/"ray tracing" or ttl/"raytracing" or spec/"raytracing")

50. There are 380 U.S. patents directed to improvements in ray tracing. For example, determine specific locations to position a ray; super sampling; using specific probability formulas; hierarchy; and secondary rays, including others.

51. The following 380 U.S. patents are directed to ray tracing, and improvements to ray tracing; and each one of these 380 U.S. patents is incorporated herein by reference (Incorporated Patents):

9,035,945	9,024,972	9,007,388	8,988,465	8,988,449	8,988,433	8,976,199	8,970,626
8,970,592	8,970,591	8,963,918	8,952,977	8,952,961	8,933,967	8,928,658	8,907,950
8,878,873	8,872,824	8,860,733	8,860,712	8,854,369	8,854,367	8,836,702	8,823,708
8,817,014	8,797,324	8,797,322	8,791,951	8,773,422	8,760,450	8,749,552	8,736,610
8,717,366	8,698,806	8,692,828	8,675,022	8,665,271	8,659,591	8,638,332	8,629,881
8,619,094	8,619,079	8,619,078	8,593,459	8,587,588	8,581,926	8,570,322	8,564,589
8,553,028	8,547,374	8,542,231	8,520,021	8,502,819	8,493,383	8,482,561	8,466,919
8,441,482	8,436,852	8,421,821	8,421,801	8,417,261	8,411,088	8,390,618	8,379,030
8,379,026	8,379,022	8,373,715	8,373,699	8,368,694	8,363,053	8,358,305	8,355,019
8,350,846	8,339,398	8,319,825	8,310,481	8,300,049	8,284,195	8,275,397	8,274,530
8,269,770	8,259,105	8,259,101	8,253,753	8,248,416	8,248,415	8,248,412	8,248,405
8,248,401	8,243,073	8,237,730	8,237,711	8,218,903	8,217,931	8,212,816	8,212,806
8,207,968	8,203,559	8,189,006	8,189,003	8,189,001	8,188,997	8,188,996	8,179,566
8,164,590	8,160,391	8,159,499	8,159,492	8,139,060	8,134,556	8,134,551	8,130,244
8,120,991	8,120,609	8,115,763	8,106,921	8,106,906	8,102,391	8,089,481	8,085,267
8,081,185	8,077,183	8,072,454	8,063,902	8,049,752	8,035,641	8,031,210	8,031,191
8,026,915	8,018,457	8,013,857	8,009,176	7,991,240	7,983,788	7,978,192	7,973,790
7,969,433	7,952,583	7,952,574	7,940,266	7,932,913	7,932,905	7,924,295	7,903,113
7,884,819	7,880,743	7,864,187	7,864,174	7,852,336	7,830,379	7,808,501	7,808,500

1	7,796,128	7,791,602	7,773,087	7,768,524	7,755,628	7,755,627	7,737,974	7,737,970
2	7,719,544	7,719,532	7,710,431	7,692,647	7,688,320	7,652,666	7,619,626	7,609,264
3	7,593,019	7,589,729	7,586,489	7,573,475	7,554,540	7,548,238	7,542,044	7,525,543
4	7,515,152	7,499,053	7,495,664	7,479,962	7,479,960	7,471,301	7,456,837	7,446,777
5	7,439,973	7,432,935	7,427,996	7,414,624	7,379,060	7,358,971	7,345,687	7,324,116
6	7,321,370	7,310,098	7,289,119	7,286,971	7,268,789	7,256,782	7,250,948	7,246,045
7	7,245,301	7,233,337	7,230,624	7,230,623	7,227,555	7,218,322	7,212,207	7,199,795
8	7,196,704	7,184,042	7,173,622	7,170,510	7,167,177	7,154,504	7,148,891	7,136,790
9	7,133,044	7,133,041	7,129,944	7,129,942	7,126,605	7,123,259	7,113,184	7,106,325
10	7,102,636	7,098,915	7,084,871	7,079,157	7,079,139	7,071,938	7,071,936	7,050,054
11	7,050,053	7,047,014	7,046,243	7,034,825	7,034,818	7,027,046	7,012,615	7,012,604
12	7,002,589	7,002,570	6,999,096	6,989,832	6,985,240	6,983,082	6,982,714	6,979,084
13	6,972,758	6,961,058	6,956,570	6,943,805	6,943,789	6,940,529	6,940,508	6,933,939
14	6,924,816	6,922,193	6,919,909	6,909,436	6,864,890	6,828,978	6,825,851	6,798,409
15	6,791,567	6,788,304	6,784,882	6,781,598	6,771,272	6,753,878	6,731,304	6,731,284
16	6,724,393	6,724,384	6,704,017	6,697,062	6,646,640	6,639,597	6,628,298	6,597,359
17	6,583,787	6,570,578	6,567,083	6,556,200	6,515,664	6,512,995	6,496,597	6,466,227
18	6,466,207	6,437,796	6,434,278	6,429,864	6,421,050	6,414,684	6,414,681	6,400,365
19	6,400,364	6,373,485	6,369,818	6,359,629	6,348,919	6,342,889	6,329,989	6,329,988
20	6,324,347	6,323,863	6,307,568	6,300,965	6,285,376	6,268,863	6,226,005	6,222,937
21	6,157,387	6,157,385	6,128,021	6,111,582	6,097,854	6,097,394	6,064,393	6,061,065
22	6,044,181	6,034,691	6,016,150	6,009,190	5,987,164	5,986,668	5,966,134	5,966,131
23	5,940,067	5,936,630	5,933,146	5,903,274	5,823,780	5,821,942	5,809,219	5,796,407
24	5,742,796	5,742,293	5,729,672	5,717,848	5,715,384	5,687,307	5,684,937	5,673,376
25	5,638,499	5,602,979	5,594,854	5,594,850	5,594,844	5,588,098	5,583,975	5,566,283
26	5,553,214	5,550,959	5,548,693	5,528,741	5,528,737	5,526,471	5,488,700	5,384,901
27	5,384,899	5,371,778	5,355,442	5,313,568	5,305,430	5,299,298	5,297,043	5,283,859
28	5,257,355	5,239,624	5,138,699	5,058,042	5,038,302	5,031,117	5,025,400	4,987,554

4,928,250 4,865,423 4,807,158 4,645,459

CAD Surface Gradients 1970 through 2010

52. For more than 40 years, starting in about the year 1970 through about 2010, CAD programs used ray tracing and a series of flat polygons to approximate curved surfaces. (polygon approximation). For example, polygon approximation uses a specific quantity of triangular surface area to define one portion of a curved surface. Polygon approximation does not create realistic surfaces. Polygon approximation is inherently described in the Incorporated Patents.

53. Vector Plane intersection equations do not exist prior to Coffelt's Vector Work. These Incorporated Patents are a basis. Prior CAD explicitly use ray tracing.

54. Polygon approximation surfaces are not realistic. Therefore, realistic surface gradients can not exist on these surface approximations. For more than 40 years, starting in about the year 1970 through about 2010, all CAD surface gradients are non-realistic approximations. *See* EXHIBIT 137. The AutoCAD drawing in EXHIBIT 137 is exemplary of all prior CAD non-realistic surface shading.

55. Polygon approximation surfaces are not realistic. Therefore, realistic shadows can not exist on these surface approximations. For more than 40 years, starting in about the year 1970 through about 2010, all CAD shadows are non-realistic approximations. *See* United States District Court, for the Central District of California, case No. ED CV16-00457 Coffelt v Nvidia, doc No. 38, doc No. 41 which is incorporated herein by reference; *See* United States Court of Appeals for the Federal Circuit case No. 17-1119 Document 2, Filed: 11/08/2016 which is incorporated herein by reference; *See* United States Court of Appeals for the Federal Circuit, Coffelt v. Nvidia, case No. 17-1119 Document 21, which is incorporated herein by reference.

Coffelt's Photorealistic CAD Surface Gradients

56. Coffelt's CAD Work creates results which are significantly distinct from all prior CAD results. Coffelt's CAD Work uses Coffelt's Vector Work. Coffelt's Vector Work creates photorealistic surfaces. *See* EXHIBIT 135 and EXHIBIT 136. Coffelt's CAD Work creates photorealistic surface

gradients. *See* EXHIBIT 135 and EXHIBIT 136. All prior CAD programs create non-realistic gradient approximations. *See* EXHIBIT 137

57. The concept of Coffelt's Gradient Work is explained in EXHIBIT 106. In EXHIBIT 106, page 1, shows: a graphic surface (S); a reflection vector (rfla); a reflection vector (rflb); a view point VP; a light source point SP; a maximum distance to the View Point (d0b); a minimum distance to the View Point d0a; and a surface normal N. EXHIBIT 106 shows a core component of Coffelt's Gradient Work which is the surface gradient is based on the angle between the reflection vector and the direction of view. There are 2 directions of view in EXHIBIT 106, view vector (vpa), and view vector (vpb). There are 2 angles in EXHIBIT 106 used to derive the photorealistic gradient. Angle (a) is the angle between vector (rfla) and vector (vpa); (a) is the maximum angle for all reflection vectors on surface (S). Angle (b) is the angle between vector (rflb) and vector (vpb); (b) is the minimum angle for all reflection vectors on surface (S).

58. EXHIBIT 106, page 2, shows a linear equation used to derive the photorealistic surface gradient. The quantity of color shift is derived by the linear equation shown on page 2 of EXHIBIT 106. In this example, (-50) is the maximum color shift, and zero is the minimum color shift. The maximum color shift occurs at (maxd); and the minimum shift occurs at (mind). The point slope equation is derived from the given values of : (-50), (mind), and (maxd). During runtime of Coffelt's Gradient Work, many various angles will be derived between the view direction and the reflection vector. The quantity of color shift is derived by this linear equation for each distinct d0 (or cosine of the angle), or other equivalent parameter.

Larry Gritz, Open Source Shading Language

59. Open Source Shading Language (OSL) by Larry Gritz (Gritz) is allegedly a new programming language. A review of OSL source code shows OSL is merely a C++ language Application Program Interface (API). A unique variable name "closure" is purportedly the basis for OSL being a new language. However, Gritz does not provided any technical explanation of how OSL is a new language. Gritz only discloses that a "closure" is new, without any technical explanation of the physical structure of a "closure".

60. OSL creates photorealistic surfaces in CAD. OSL creates results identical to Coffelt's CAD Work results. See EXHIBIT 120, EXHIBIT 121, EXHIBIT 122, EXHIBIT 123, EXHIBIT 124, EXHIBIT 125

61. Gritz explains that it is optimal for Sony ImageWorks to give away valuable software free, rather than keep it to themselves. Gritz also explains that the programing language C is "clunky", and is the cause for the necessity for a new programing language. Gritz does not explain the meaning of "clunky"; and does not explain how he has overcome the problem of "clunky". Gritz merely explains the awesome results of his allegedly new programming language. Portions of OSL source code is attached in EXHIBIT 115. This OSL source code is identical to Coffelt's CAD Work or is a derivative work of Coffelt's CAD Work. This OSL source code is identical to Coffelt's Gradient Work or is a derivative work of Coffelt's Gradient Work. *See* EXHIBIT 115, and EXHIBIT 104. A comparison of these 2 exhibits 115 and 104 shows that all components of Coffelt's copyrighted work exists in OSL. Furthermore, the results of OSL are identical to Coffelt's copyrighted work results.

62. Larry Gritz is awarded a technical achievement award for OSL, without any any explanation of his technical achievement. Larry Gritz receives an Academy Award for only the photorealistic results. *See* EXHIBIT 112, and EXHIBIT 113

63. Blender publications confirm that OSL is based on Coffelt's Gradient Work. *See* EXHIBIT 114 In EXHIBIT 114, the Blender publication alleges that the photorealistic surface gradient is based on the direction of view, and the reflection vector. This basis for OSL is identical to Coffelt's concept for Coffelt's Gradient Work. *See* EXHIBIT 106, EXHIBIT 114, EXHIBIT 102, EXHIBIT 103, EXHIBIT 104

64. A BusinessWire publication confirms that OSL is used in Autodesk Beast. *See* EXHIBIT 116

65. A Sony Imageworks publication confirms that OSL is used in Autodesk Beast. *See* EXHIBIT 117

Autodesk

66. Autodesk's products AutoCad, Fusion 360, Maya, InfraWorks, AutoCAD Civil 3D, Revit,

and Beast (Autodesk Products), is essentially identical to Coffelt's Work and clearly used Coffelt's Work as its basis.

67. The following individuals have been served with a Cease and Desist letter from Coffelt: Autodesk Executives:

Andrew Anagnost, Carl Bass, Crawford W. Beveridge, Steve Blum, Chris Bradshaw, Moonhie Chin, Pascal W. DiFronzo, Reid French, Thomas Georgens, R. Scott Herren, Richard S. Hill, Jeff Kowalski, Mary T. McDowell, Lorrie M. Norrington, Elizabeth Rafael, Stacy J. Smith, Eric Mitchel, Will Harris, Jorge Garcia, Edwin Robledo;

68. The following GitHub individuals have been served with a Cease and Desist letter from Coffelt:

Alison Marcozzi, Chris Wanstrath (Corporate Executive Officer).

69. Autodesk publications confirm that the above identified Autodesk products use OSL *See* EXHIBIT 118, and EXHIBIT 119

70. The above identified Autodesk Products contain a copy of OSL. The above identified Autodesk Products is a derivative work of OSL. The above identified Autodesk products is a distribution of OSL. Larry Gritz is not authorized to copy Coffelt's copyrighted works. Larry Gritz is not authorized to make derivative works of Coffelt's copyrighted works. Larry Gritz is not authorized to distribute Coffelt's copyrighted works.

71. Coffelt has Not authorized any rights, in Coffelt's copyrighted works.

72. Coffelt has Not authorized any title, in Coffelt's copyrighted works.

73. Coffelt has Not authorized any interest, in Coffelt's copyrighted works.

74. For the above reasons, The above identified Autodesk Products is an unauthorized copy of Coffelt's copyrighted works.

75. For the above reasons, The above identified Autodesk Products is an unauthorized derivative work of Coffelt's copyrighted works.

76. For the above reasons, The above identified Autodesk Products is an unauthorized distribution of Coffelt's copyrighted works.

Autodesk Access To Coffelt's Copyrighted Work

77. Autodesk attained access to Coffelt's copyrighted works on February 28, 2013 by Coffelt's U.S patent No. 8,614,710 publication.

78. On about March 18, 2010, at 428 Devener Street, Apt. # C, Riverside, CA 92507 California Department of Corrections (CDC) agents forcefully took copies of Coffelt's Work. Autodesk has a significant relationship with CDC. Evidence of this significant relationship will be provided to this Court.

79. In about the year 2011, at 1195 Spring Street, Apt. # C Riverside, CA 92507 CDC agents forcefully took copies of Coffelt's Work. Evidence of this unauthorized copy of Coffelt's copyrighted work will be provided to this Court.

80. In about the year 2013, at 14327 Frederick Street, Moreno Valley, CA 92553 CDC agents forcefully took copies of Coffelt's Work. Evidence of this unauthorized copy of Coffelt's copyrighted work will be provided to this Court.

81. For the above reasons, Autodesk attained access to Coffelt's copyrighted works on at least 3 occurrence, first in the year 2010, second about the year 2011, and third, about the year 2013.

FIRST CAUSE OF ACTION

(Copyright Infringement – 17 U.S.C. §501)

82. Plaintiff repeats and incorporates by this reference the allegations set forth in paragraphs 1 through 81, inclusive.

83. Plaintiff Coffelt is the author and sole owner of all rights title and interest of the claimed works derived, reproduced, and distributed by Autodesk through various products including without limitation, AutoCad, Fusion 360, Maya, InfraWorks, AutoCAD Civil 3D, Revit.

84. For each of the claimed works in this matter, Plaintiff holds a copyright registration certificate from the United States Copyright Office.

85. Without authorization, Autodesk copied, derived, reproduced, and distributed the following Plaintiff owned and copyrighted claimed work including:

"emoshaGraphics CAD alpha" registration No. TXu002037997

86. Through their conduct averred herein, Defendants have infringed Plaintiffs' copyright in the above identified Autodesk Products, in violation of Section 501 of the Copyright Act, 17 U.S.C. § 501(a).

87. Defendants' acts of infringement are willful, intentional and purposeful, in disregard of and with indifference to Plaintiff's rights.

88. As a direct and proximate result of said infringement by Defendants, Plaintiff is entitled to damages of at least $33,000,000,000 to be proven at trial.

89. Plaintiff is also entitled to Defendant's profits attributable to the infringement, pursuant to 17 U.S.C. § 504(b), including an accounting of such profits.

90. Plaintiff is further are entitled to Plaintiff's attorneys' fees and full costs pursuant to 17 U.S.C. § 505 and otherwise according to law.

91. As a direct and proximate result of the foregoing acts and conduct, Plaintiff has sustained and will continue to sustain substantial, immediate, and irreparable injury, for which there is no adequate remedy at law. Plaintiff is informed and believe and on that basis aver that unless enjoined and restrained by this Court, Defendants will continue to infringe Plaintiff's rights in the Infringed Works. Plaintiff is entitled to preliminary and permanent injunctive relief to restrain and enjoin Defendants' continuing infringing conduct.

RELIEF

92. WHEREFORE, Plaintiff request the following judgement against Defendant Autodesk as follows:

93. For Plaintiff's damages in the amount of $ 33,000,000,000 (thirty three billion dollars) and any additional damages proven at trial; and Defendant's profits;

94. For preliminary and permanent injunction enjoining Defendant Autodesk and all persons acting in concert or participation with Autodesk from (a) directly or indirectly reproducing, distributing, or otherwise infringing in any manner on Plaintiff's copyrighted works.

95. For Plaintiff's attorneys' fees and full costs incurred in this action.

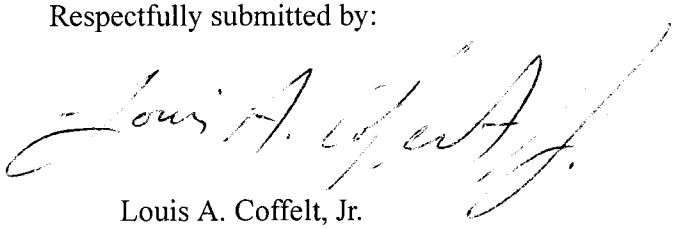
96. For any additional relief as this Court may deem just and proper.

DEMAND FOR JURY TRIAL

Plaintiff Coffelt hereby request a jury trial for all issues raised in this action.

Date: August 21, 2017

Respectfully submitted by:



Louis A. Coffelt, Jr.
email: Louis.Coffelt@gmail.com
231 E. Alessandro Blvd., Ste 6A-504
Riverside, CA 92508
Phone: (951) 790-6086
Pro Se

original

APPENDIX

EXHIBIT 100

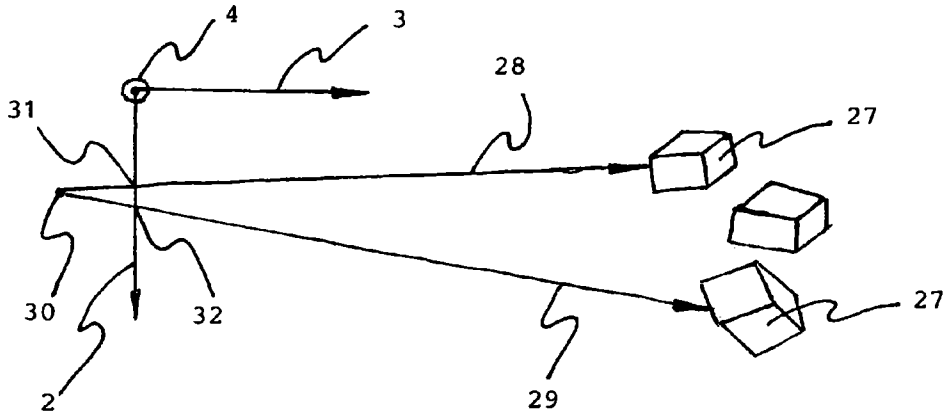


FIG. 7

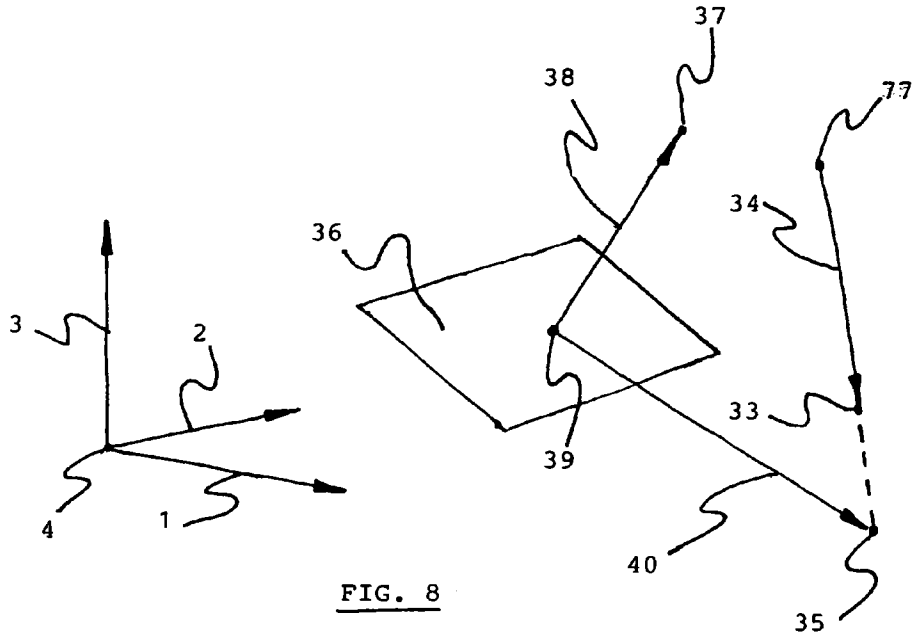


FIG. 8

columnD=pti*pixelsPerInchD; column=unsigned int(columnD); Indexx=row*bitmapPixelWidth+column; priorlength=lengthv[Indexx]; currentlength=sqrt (vppti*vppti+vpptj*vpptj+vpptk*vpptk); if(currentlength<priorlength) <[lengthv[Indexx]=currentlength;]> pti+=0.001; count1++;]> while(count2<700) <[ptj=m0*pti−4.3; vppti=vpi−pti; vpptj=vpj−ptj; vpptk=vpk−ptk; IntersectVectorWithPlane(intpti, intptj, intptk, vpi, vpj, vpk, pti, ptj, ptk, N1*i*, N1*j*, N1*k*, N0*i*, N0*j*, N0*k*); if(intpti<0.0 or intpti> bitmapWidthInches or intptj<0.0 or intptj> bitmapHeightInches)<[pti+=0.001; count2++; if(count2<700)<[continue;]> else <[break;]>]> rowD=ptj*pixelsPerInchD; row=unsigned int(rowD); columnD=pti* pixelsPerInchD; column=unsigned int(columnD); Indexx=row* bitmapPixelWidth+column; priorlength=lengthv[Indexx]; currentlength=sqrt (vppti*vppti+vpptj*vpptj+vpptk*vpptk); difference1=abs (currentlength−priorlength); if(difference1<0.0001)< [pointIsVisible=true; red=240; green=0; blue=0; bitmapx.SetPixel(row, column, red, green, blue);]> else <[pointIsVisible=false;]> pti+=0.001; count2++;]> while (count3<900)<[ptj=m1*pti−2.89; vppti=vpi−pti; vpptj=vpj−ptj; vpptk=vpk−ptk; IntersectVectorWithPlane(intpti, intptj, intptk, vpi, vpj, vpk, pti, ptj, ptk, N1*i*, N1*j*, N1*k*, N0*i*, N0*j*, N0*k*); if(intpti<0.0 or intpti> bitmapWidthInches or intptj<0.0 or intptj> bitmapHeightInches)<[pti+=0.001; count3++; if(count3<900)<[continue;]> else<[break;]>]> rowD=ptj*pixelsPerInchD; row=unsigned int(rowD); columnD=pti*pixelsPerInchD; column=unsigned int(columnD); Indexx=row*bitmapPixelWidth+column; priorlength=lengthv[Indexx]; currentlength=sqrt (vppti*vppti+vpptj*vpptj+vpptk*vpptk); difference1=abs (currentlength−priorlength); if(difference1<0.0001)< [pointIsVisible=true; red=0; green=0; blue=250; bitmapx.SetPixel(row, column, red, green, blue);]> else <[pointIsVisible=false;]> pti+=0.001; count3++;]>]>

FIG. **8** shows a perspective view of a vector (**34**) intersecting a plane (**36**) at point (**35**). A normal vector (**38**) of plane (**36**) is shown. The normal vector is formed by point N**0** (**39**) and point N**1** (**37**). Point N (**39**) is on plane (**36**). Vector (**34**) is formed of any two points, point (**77**) and point (**33**). Vector (**40**) is formed of two points, point (**35**) and point (**39**). Vector (**40**) is in plane (**36**). The following c++ code shows an example of 'Vector Plane Intersection', and is the function used above. where, intpti, intptj, intptk is intersection point (**35**); pt**1** is point (**77**); pt**0** is point (**33**); N**0** is point (**39**); N**1** is point (**37**):

c++ <[void IntersectVectorWithPlane(double& intptiP, double& intptjP, double& intptkP, double pt1*i*, double pt1*j*, double pt1*k*, double pt0*i*, double pt0*j*, double pt0*k*, double N1*i*P, double N1*j*P, double N1*k*P, double N0*i*P, double N0*j*P, double N0*k*P)<[double Ni=N1*i*P−N0*i*P; double Nj=N1*j*P−N0*j*P; double Nk=N1*k*P−N0*k*P; double testdenom=abs(pt1*i*−pt0*i*); if(testdenom<1.0e−9)<[return;]> double mji=(pt1*j*−pt0*j*)/(pt1*i*−pt0*i*); double mki=(pt1*k*=pt0*k*)/(pt1*i*−pt0*i*); testdenom=abs(Ni+Nj*mji+Nk*mki); if(testdenom<1.0e−9) <[return;]> tempi=(N0*i*P*Ni+Nj*mji*pt0*i*−Nj*pt0*j*+ Nj*N0*j*P+Nk*mki*pt0*i*−Nk*pt0*k*+Nk*N0*k*P)/(Ni+Nj*mji+ Nk*mki); intptiP=tempi; intptjP=mji*(tempi−pt0*i*)+pt0*j*; intptkP=mki*(tempi−pt0*i*)+pt0*k*;]>]>

The general formula for intptiP is formed by combining a general formula for a plane with a general formula for a line. More specifically, the projection of a vector to the i-j plane, and the projection of the vector to the i-k plane. The general formula for a plane is set in an equation that any vector in the plane dotted with the planes normal vector is zero. N dot v is zero. where N is the normal vector of the plane, and v is any vector in the plane. A general formula for a line is: j=mji*(i−i**0**)+j**0** where i**0**, j**0**, and mji are given. Also, k=mki*(i−i**0**)+k**0** where i**0**, k**0**, and mki are given. These two equations for a line are substituted in the equation for a plane; and solved for i. This substitution eliminates variables j and k. Only i remains in the equation. Solving for i yields the equation above in the c++code snippet.

The graphic object structure analysis may also include reflection vectors. A reflection vector can be derived for any point in a graphic object. This reflection vector may intersect any graphic object. e.g. any plane, sphere, or surface. The intersection point can be assigned any selected pixel color. For example, a light source contacts a particular point on a blue surface; a reflection vector is calculated at this particular point; the reflection vector intersects a red sphere; the intersection point is set to blue. The following c++code snippet sets forth an example to calculate a reflection vector.

c++ <[void ReflectionVector(double& rpti, double & rptj, double& rptk, double Ni, double Nj, double Nk, double s1*i*, double s1*j*, double s1*k*, double ai, double aj, double ak)< [double si=s1*i*−ai; double sj=s1*j*−aj; double sk=s1*k*−ak; double lengths=sqrt(si*si+sj*sj+sk*sk); double lengthN=sqrt(Ni*Ni+Nj*Nj+Nk*Nk); double NdotS= (Ni*si+Nj*sj+Nk*sk)/(lengthN*lengths); double testDot=abs(NdotS); if(testDot<1.0e−6)<[return;]> else if(testDot>0.9999)<[rpti=s1*i*; rptj=s1*j*; rptk=s1*k*; return;]> double phi=a cos(NdotS); double Nri=0.0; double Nrj=0.0; double Nrk=0.0; double rxi=0.0; double rxj=0.0; double rxk=0.0; double tlen=2.0*lengths*sin(phi); Nri=Nj*sk− sj*Nk; Nrj=−(Ni*sk−si*Nk); Nrk=Ni*sj−si*Nj; rxi=Nj*Nrk−Nrj*Nk; rxj=−(Ni*Nrk−Nri*Nk); rxk=Ni*Nrj−Nri*Nj; double lengthrx=sqrt(rxi*rxi+rxj*rxj+ rxk*rxk); if(lengthrx>1.0e−6)<[cx=tlen/lengthrx;]> else <[return;]> double tri=cx*rxi; double trj=cx*rxj; double trk=cx*rxk; rpti=s1*i*+tri; rptj=s1*j*+trj; rtpk=s1*k*+trk;]>]>

In the above example for the reflection vector, The tail end of the reflection vector is ai, aj, ak; the terminal end of the reflection vector is rpti, rptj, rptk. The reflection vector is: (rpti−ai)i+(rptj−aj)j+(rptk−ak)k; ai, aj, ak, the intersection point of a light source vector with the plane. Ni, Nj, Nk is the normal vector of the plane.

A graphic object structure may also include translucent surfaces. A translucent surface can be derived by intersecting a vector with one or more surfaces. Next, set the selected pixel color at a relatively less pixel per inch resolution. For example, for a bitmap having a resolution of 1000 pixels per inch, a translucent surface can be attained by setting a background image at about 500 pixels per inch. For example, a foreground rectangular translucent blue surface; and a background linear red surface; initially all pixels in the bitmat are blue; Next, the program iterates thru the equation of the line; and assign a red pixel at a density of 500 pixels per inch.

One method to create a translucent surface is using a 'next least length' concept. This concept is related to a 'visible' point on the geometric object. Opaque surfaces described above, have only one point per steradian which is 'visible'. In comparison, for a translucent surface, there may be to or three or more points in one particular steradian which are 'visible'. e.g. the visible points are 2 points having the least position vector length. The following is an example c++ code snippet showing a method to calculate a translucent surface:

The following is a summary of c++ code for translucent surface calculation: This code may be set in line with the above c++ code examples; c++ <vector> priorLength**0**, <vector> priorLength**1**, and <vector> priorLength**2** contain values having a respective next least length; for example, at index **33**, priorLength**0**[**33**]==22.15; priorLength**1**[**33**]==28.76; prior-

EXHIBIT 101

```
void ImageAndPanelCls::IntersectijPlane(double &scrnxP, double &scrnyP, double ptxP, double ptyP, double ptzP,
double rptxP, double rptyP, double rptzP)
{
   si = ptxP - rptxP;
   sj = ptyP - rptyP;
   sk = ptzP - rptzP;
   i = abs(si);
   j = abs(sj);
   k = abs(sk);
   s0i = ptxP;
   s0j = ptyP;
   s0k = ptzP;
   if (i > 0.000000001)
   {
      mji = sj / si;
      mki = sk / si;
      tempi = (mki * s0i - s0k) / mki;
      tempj = mji * (tempi - s0i) + s0j;
   }
   else if (j > 0.000000001)
   {
      mij = si / sj;
      mkj = sk / sj;
      tempj = (mkj * s0j - s0k) / mkj;
      tempi = mij * (tempj - s0j) + s0i;
   }
   else if (k > 0.000000001)
   {
      mik = si / sk;
      mjk = sj / sk;
      tempi = mik * (-s0k) + s0i;
      tempj = mjk * (-s0k) + s0j;
   }
      scrnxP = tempi;
      scrnyP = tempj;
}
//
void ImageAndPanelCls::IntersectjkPlanePartialSolution(double &jP, double &kP, double rptxP, double rptyP, double
rptzP, double ptxP, double ptyP, double ptzP)
{
   si = rptxP - ptxP;
   sj = rptyP - ptyP;
   sk = rptzP - ptzP;
      //
   k = abs(sk);
      //
      s0i = ptxP;
      s0j = ptyP;
   s0k = ptzP;
   if (k > 0.000000001)
   {
      mik = si  / sk;
      mjk = sj / sk;
```

```
      tempk = (mik * s0k - s0i) / mik;
      tempj = mjk * (tempk - s0k) + s0j;
   }
      jP = tempj;
      kP = tempk;
}
//
void ImageAndPanelCls::IntersectAnyPlanePartialSolution(double &ixintersectionP, double &jxintersectionP, double
&kxintersectionP, double N1iP, double N1jP, double N1kP, double N0iP, double N0jP, double N0kP, double ptxP,
double ptyP, double ptzP, double rptxP, double rptyP, double rptzP)
{
      N1ic = N1iP;
      N1jc = N1jP;
      N1kc = N1kP;
      N0ic = N0iP;
      N0jc = N0jP;
      N0kc = N0kP;
      Nic = N1ic - N0ic;
      Njc = N1jc - N0jc;
      Nkc = N1kc - N0kc;
   si = ptxP - rptxP;
   sj = ptyP - rptyP;
   sk = ptzP - rptzP;
   i = abs(si);
   j = abs(sj);
   k = abs(sk);
   s0i = ptxP;
   s0j = ptyP;
   s0k = ptzP;
   if (k > 0.000000001)
   {
      mik = si / sk;
      mjk = sj / sk;
      tempk = (mik * s0k * Nic - s0i * Nic + N0ic * Nic + mjk * s0k * Njc - s0j * Njc + N0jc * Njc + N0kc * Nkc) /
(mik * Nic + mjk * Njc + Nkc);
      tempi = mik * (tempk - s0k) + s0i;
            tempj = mjk * (tempk - s0k) + s0j;
   }
      ixintersectionP = tempi;
      jxintersectionP = tempj;
      kxintersectionP = tempk;
}
//
```

EXHIBIT 102

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace VecPlnInt
{
    class ColorByReflectionVec
    {
        double ColorDouble = 0.0;
        int ColorInt = 0;
        //
        public void SetColorByReflectionVec(ref int RedComponentP, ref int GreenComponentP
    , ref int BlueComponentP,ref int TotalChanges, double AdotBp1, int BaseColorRedP, int
    BaseColorGreenP, int BaseColorBlueP)
        {
            // turquoise is approx 2f f2 f1
            // decreasing blue shifts towards green
            // decreasing green shifts towards royal blue (dark)
            // use 2 points of Reflection vector, and 2 points (view pt, surface pt)
    vector, cos of angle between these two vectors to set color
            // if adotb < 0 : darken color
            // total shift is  3 red w/ 16  blue
            //                 3 blue w/ 16 red
            //                 3 green w/ 16 blue
            //                 3 blue w/ 16 green
            //ShiftLimit1 = 255 - ShiftNum01;
            //ShiftLimit2 = 255 - ShiftNum02;
            //ShiftLimit3 = 255 - ShiftNum03;
            if (AdotBp1 > 0)
            {
                ColorDouble = 224 * AdotBp1;
                ColorInt = (int)ColorDouble;
                GreenComponentP = ColorInt;
            }
            else
            {
                GreenComponentP = 0;
            }
            RedComponentP = BaseColorGreenP;
            BlueComponentP = BaseColorBlueP;
            TotalChanges++;
        }
    }
}
```

EXHIBIT 103

103-1

```
#include "StdAfx.h"
#include "Objects_Cls.h"
#include <cmath>

void Objects_Cls::Sphere_____________Steradians__(cli::array<double, 1>^ &StrDistV01p, cli::array<double, 1>^
&VisibleV01p, int systemNum01P, int numSurface01P, double t0p)
{
	__int64 SizeSTRv = StrDistV01p->Length;
	__int64 SizeVISv = VisibleV01p->Length;
	short int systemNum = systemNum01P;
	short int numSurface = numSurface01P;
	t0 = t0p;
	SetDynamicData(systemNum);
	double lampSphereR = 0.4;
	double mainSphereR = 0.5;
	double ptx00a = 0.0;
	double pty00a = 0.0;
	double ptz00a = 0.0;
	double ptx02a = 0.0;
	double pty02a = 0.0;
	double ptz02a = 0.0;
	int scrnindx = 0;
	int scrncolx = 0;
	int scrnrowx = 0;
	double scrncolxD = 0.0;
	double scrnrowxD = 0.0;
	double scrnxA = 0.0;
	double scrnyA = 0.0;
	double scrnxC = 0.0;
	double scrnyC = 0.0;
	double cVisDistA = 0.0;
	double pVisDistA = 0.0;
	int StrIndxPxA = 0;
	double cStrDistA = 0.0;
	double pStrDistA = 0.0;
	//
	double d0 = 0.0;
	double N0000x = 0.0;
	double N0000y = 0.0;
	double N0000z = 0.0;
	double N0100x = 0.0;
	double N0100y = 0.0;
	double N0100z = 0.0;
	double N00x = 0.0;
	double N00y = 0.0;
	double N00z = 0.0;
	double N0002x = 0.0;
	double N0002y = 0.0;
	double N0002z = 0.0;
	double N0102x = 0.0;
	double N0102y = 1.0;
	double N0102z = 0.0;
	NextCoordinatesType3conversionOnly(N0000x, N0000y, N0000z, N0002x, N0002y, N0002z);
```

```
NextCoordinatesType3conversionOnly(N0100x, N0100y, N0100z, N0102x, N0102y, N0102z);
N00x = N0100x - N0000x;
N00y = N0100y - N0000y;
N00z = N0100z - N0000z;
double spax = 0.0;
double spay = 0.0;
double spaz = 0.0;
double spdx = 0.0;
double spdy = 0.0;
double spdz = 0.0;
double vpax = 0.0;
double vpay = 0.0;
double vpaz = 0.0;
double bx = 0.0;
double by = 0.0;
double bz = 0.0;
double cx = 0.0;
double cy = 0.0;
double cz = 0.0;
double spacos = 1.0;
double spasin = 1.0;
double dx = 0.0;
double dy = 0.0;
double dz = 0.0;
double ex = 0.0;
double ey = 0.0;
double ez = 0.0;
double rptx = 0.0;
double rpty = 0.0;
double rptz = 0.0;
double rflx = 0.0;
double rfly = 0.0;
double rflz = 0.0;
double spadotc = 1.0;
double vpdotrfl = 1.0;
double phi = 0.0;
double phix = 0.0;
double thetax = 0.0;
double theta = 0.0;
double c0 = 1.0;
double lenc = 1.0;
double lend = 1.0;
double lene = 1.0;
double lenspa = 1.0;
double lenvpa = 1.0;
double lenrfl = 1.0;
//
double pi = 3.1415926;
double twopi = 2.0 * pi;
double pid2 = pi / 2.0;
double threepid2 = 3.0 * pi / 2.0;
double pid6 = pi / 6.0;
double fivepid6 = 5.0 * pi / 6.0;
double sevenpid6 = 7.0 * pi / 6.0;
```

```
double elevenpid6 = 11.0 * pi / 6.0;
double r0 = 1.5;
double lenxy = 0.07;
double xz_limit = 0.0;
double m_s = 0.0;
double x = 0.0;
double y = 0.0;
double z = 0.0;
if (systemNum == 850 || systemNum == 851 || systemNum == 852 || systemNum == 853)
{
    dphi_c = dphi_850;
}
while (phix < twopi)
{
    thetax = theta_i;
    while (thetax < twopi)
    {
        pty02a = r_sphere_ui * cos(thetax);
        ptz02a = r_sphere_ui * sin(thetax) * sin(phix);
        ptx02a = r_sphere_ui * sin(thetax) * cos(phix);
        N0102x = ptx02a;
        N0102y = pty02a;
        N0102z = ptz02a;
        NextCoordinatesType3conversionOnly(N0100x, N0100y, N0100z, N0102x, N0102y, N0102z);
        N00x = N0100x - N0000x;
        N00y = N0100y - N0000y;
        N00z = N0100z - N0000z;
        NextCoordinatesType3(ptx00a, pty00a, ptz00a, scrnxA, scrnyA, ptx02a, pty02a, ptz02a);
        if (scrnxA > 0.0 && scrnxA < scrnWinches && scrnyA > 0.0 && scrnyA < scrnHinches)
        {
            scrncolxD = scrnxA * scrnppiD;
            scrnrowxD = scrnyA * scrnppiD;
            scrncolx = int(scrncolxD);
            scrnrowx = int(scrnrowxD);
            scrnindx = scrnrowx * scrnWpx + scrncolx;
            if (scrnindx < SizeVISv)
            {
                pVisDistA = VisibleV01p[scrnindx];
                cVisDistA = sqrt(si * si + sj * sj + sk * sk);
                if (cVisDistA < pVisDistA)
                {
                    VisibleV01p[scrnindx] = cVisDistA;
                    //
                    spax = spx - ptx00a;
                    spay = spy - pty00a;
                    spaz = spz - ptz00a;
                    lenspa = sqrt(spax * spax + spay * spay + spaz * spaz);
                    vpax = vpx - ptx00a;
                    vpay = vpy - pty00a;
                    vpaz = vpz - ptz00a;
                    lenvpa = sqrt(vpax * vpax + vpay * vpay + vpaz * vpaz);
                    bx = N00y * spaz - spay * N00z;
                    by = -(N00x * spaz - spax * N00z);
                    bz = N00x * spay - spax * N00y;
```

```
cx = N00y * bz - by * N00z;
cy = -(N00x * bz - bx * N00z);
cz = N00x * by - bx * N00y;
lenc = sqrt(cx * cx + cy * cy + cz * cz);
spadotc = (spax * cx + spay * cy + spaz * cz) / (lenspa * lenc);
phi = acos(spadotc);
spacos = lenspa * abs(cos(phi));
spasin = lenspa * sin(phi);
c0 = spacos / lenc;
ex = c0 * cx;
ey = c0 * cy;
ez = c0 * cz;
dx = ptx00a - ex;
dy = pty00a - ey;
dz = ptz00a - ez;
spdx = spx - dx;
spdy = spy - dy;
spdz = spz - dz;
rptx = ptx00a + ex + spdx;
rpty = pty00a + ey + spdy;
rptz = ptz00a + ez + spdz;
rflx = rptx - ptx00a;
rfly = rpty - pty00a;
rflz = rptz - ptz00a;
lenrfl = sqrt(rflx * rflx + rfly * rfly + rflz * rflz);
vpdotrfl = (vpax * rflx + vpay * rfly + vpaz * rflz) / (lenvpa * lenrfl);
theta = acos(vpdotrfl);
d0 = lenvpa * sin(theta);
if (systemNum == 850 && d0 < min_d_850)
{
    min_d_850 = d0;
}
if (systemNum == 850 && d0 > max_d_850)
{
    max_d_850 = d0;
}
if (systemNum == 851 && d0 < min_d_851)
{
    min_d_851 = d0;
}
if (systemNum == 851 && d0 > max_d_851)
{
    max_d_851 = d0;
}
if (systemNum == 852 && d0 < min_d_852)
{
    min_d_852 = d0;
}
if (systemNum == 852 && d0 > max_d_852)
{
    max_d_852 = d0;
}
if (systemNum == 853 && d0 < min_d_853)
{
```

```
                                min_d_853 = d0;
                            }
                            if (systemNum == 853 && d0 > max_d_853)
                            {
                                max_d_853 = d0;
                            }
                        }
                    }
                }
                thetax += dphi_c;
            }
            phix += dphi_c;
        }
        int stophere = 0;
}
//
void Objects_Cls::Sphere_____________Iteration__(cli::array<System::Byte, 1>^ &RedVp, cli::array<System::Byte,
1>^ &GreenVp, cli::array<System::Byte, 1>^ &BlueVp, cli::array<double, 1>^ StrDistV00p, cli::array<double, 1>^
VisibleV00p, int systemNum00P, int numSurface00P, int numColor00P)
{
        __int64 sizeRedV = RedVp->Length;
        __int64 SizeSTRv = StrDistV00p->Length;
        __int64 SizeVISv = VisibleV00p->Length;
        int numColor = numColor00P;
        short int systemNum = systemNum00P;
        short int numSurface = numSurface00P;
        SetDynamicData(systemNum);
        double blueD = 180.0;
        double greenD = 180.0;
        double redD = 180.0;
        int btBlueInt = 180;
        int btGreenInt = 180;
        int btRedInt = 180;
        double shiftd = 0.0;
        double mgrad = -222.0 / 10.0;
        double ptx00b = 0.0;
        double pty00b = 0.0;
        double ptz00b = 0.0;
        double ptx02b = 0.0;
        double pty02b = 0.0;
        double ptz02b = 0.0;
        double spx00 = spx;
        double spy00 = spy;
        double spz00 = spz;
        double spx02 = 0.0;
        double spy02 = 0.0;
        double spz02 = 0.0;
        double vpx00 = vpx;
        double vpy00 = vpy;
        double vpz00 = vpz;
        double vpx02 = 0.0;
        double vpy02 = 0.0;
        double vpz02 = 0.0;
        int scrnindx = 0;
```

```
int scrncolx = 0;
int scrnrowx = 0;
double scrncolxD = 0.0;
double scrnrowxD = 0.0;
double scrnxB = 0.0;
double scrnyB = 0.0;
double cVisDistB = 0.0;
double pVisDistB = 0.0;
int StrIndxPxB = 0;
double cStrDistB = 0.0;
double pStrDistB = 0.0;
double scrnxD = 0.0;
double scrnyD = 0.0;
double cVisDistD = 0.0;
double pVisDistD = 0.0;
int StrIndxPxD = 0;
double cStrDistD = 0.0;
double pStrDistD = 0.0;
double deltaStr = 1.0;
double deltaVis = 1.0;
double pid3 = 3.1415926 / 3.0;
double d0 = 0.0;
double N0000x = 0.0;
double N0000y = 0.0;
double N0000z = 0.0;
double N0100x = 0.0;
double N0100y = 0.0;
double N0100z = 0.0;
double N00x = 0.0;
double N00y = 0.0;
double N00z = 0.0;
double N0002x = 0.0;
double N0002y = 0.0;
double N0002z = 0.0;
double N0102x = 0.0;
double N0102y = 1.0;
double N0102z = 0.0;
NextCoordinatesType3conversionOnly(N0000x, N0000y, N0000z, N0002x, N0002y, N0002z);
NextCoordinatesType3conversionOnly(N0100x, N0100y, N0100z, N0102x, N0102y, N0102z);
N00x = N0100x - N0000x;
N00y = N0100y - N0000y;
N00z = N0100z - N0000z;
double spax = 0.0;
double spay = 0.0;
double spaz = 0.0;
double spdx = 0.0;
double spdy = 0.0;
double spdz = 0.0;
double vpax = 0.0;
double vpay = 0.0;
double vpaz = 0.0;
double bx = 0.0;
double by = 0.0;
double bz = 0.0;
```

```
double cx = 0.0;
double cy = 0.0;
double cz = 0.0;
double spacos = 1.0;
double spasin = 1.0;
double dx = 0.0;
double dy = 0.0;
double dz = 0.0;
double ex = 0.0;
double ey = 0.0;
double ez = 0.0;
double rptx = 0.0;
double rpty = 0.0;
double rptz = 0.0;
double rflx = 0.0;
double rfly = 0.0;
double rflz = 0.0;
double spadotc = 1.0;
double vpdotrfl = 1.0;
double phi = 0.0;
double c0 = 1.0;
double lenc = 1.0;
double lend = 1.0;
double lene = 1.0;
double lenspa = 1.0;
double lenvpa = 1.0;
double lenrfl = 1.0;
int minRed = 1000;
int maxRed = 0;
double phix = 0.0;
double thetax = 0.0;
double theta = 0.0;
double pi = 3.1415926;
double twopi = 2.0 * pi;
double pid2 = pi / 2.0;
double threepid2 = 3.0 * pi / 2.0;
double pid6 = pi / 6.0;
double fivepid6 = 5.0 * pi / 6.0;
double sevenpid6 = 7.0 * pi / 6.0;
double elevenpid6 = 11.0 * pi / 6.0;
double r0 = 1.5;
double lenxy = 0.07;
double xz_limit = 0.0;
double m_s = 0.0;
double x = 0.0;
double y = 0.0;
double z = 0.0;
int countx = 0;
int county = 0;
System::String^ data = " ";
int maxr = 0;
int minr = 1000;
int maxg = 0;
int ming = 1000;
```

```
int maxb = 0;
int minb = 1000;
if (systemNum == 850)
{
	mgrad = -50 / (max_d_850 - min_d_850);
	dphi_c = dphi_850;
}
else if (systemNum == 851)
{
	mgrad = -50 / (max_d_851 - min_d_851);
	dphi_c = dphi_850;
}
if (systemNum == 852)
{
	mgrad = -50 / (max_d_852 - min_d_852);
	dphi_c = dphi_850;
}
else if (systemNum == 853)
{
	mgrad = -50 / (max_d_853 - min_d_853);
	dphi_c = dphi_850;
}
while (phix < twopi)
{

	thetax = theta_i;
	while (thetax < twopi)
	{
		pty02b = r_sphere_ui * cos(thetax);
		ptz02b = r_sphere_ui * sin(thetax) * sin(phix);
		ptx02b = r_sphere_ui * sin(thetax) * cos(phix);
		N0102x = ptx02b;
		N0102y = pty02b;
		N0102z = ptz02b;
		NextCoordinatesType3conversionOnly(N0100x, N0100y, N0100z, N0102x, N0102y, N0102z);
		N00x = N0100x - N0000x;
		N00y = N0100y - N0000y;
		N00z = N0100z - N0000z;
		NextCoordinatesType3(ptx00b, pty00b, ptz00b, scrnxB, scrnyB, ptx02b, pty02b, ptz02b);
		if (scrnxB > 0.0 && scrnxB < scrnWinches && scrnyB > 0.0 && scrnyB < scrnHinches)
		{
			scrncolxD = scrnxB * scrnppiD;
			scrnrowxD = scrnyB * scrnppiD;
			scrncolx = int(scrncolxD);
			scrnrowx = int(scrnrowxD);
			scrnindx = scrnrowx * scrnWpx + scrncolx;
			if (scrnindx < SizeVISv)
			{
				pVisDistB = VisibleV00p[scrnindx];
				cVisDistB = sqrt(si * si + sj * sj + sk * sk);
				deltaVis = abs(cVisDistB - pVisDistB);
				if (deltaVis < 0.001)
				{
					spax = spx - ptx00b;
```

```
spay = spy - pty00b;
spaz = spz - ptz00b;
lenspa = sqrt(spax * spax + spay * spay + spaz * spaz);
vpax = vpx - ptx00b;
vpay = vpy - pty00b;
vpaz = vpz - ptz00b;
lenvpa = sqrt(vpax * vpax + vpay * vpay + vpaz * vpaz);
bx = N00y * spaz - spay * N00z;
by = -(N00x * spaz - spax * N00z);
bz = N00x * spay - spax * N00y;
cx = N00y * bz - by * N00z;
cy = -(N00x * bz - bx * N00z);
cz = N00x * by - bx * N00y;
lenc = sqrt(cx * cx + cy * cy + cz * cz);
spadotc = (spax * cx + spay * cy + spaz * cz) / (lenspa * lenc);
phi = acos(spadotc);
spacos = lenspa * abs(cos(phi));
spasin = lenspa * sin(phi);
c0 = spacos / lenc;
ex = c0 * cx;
ey = c0 * cy;
ez = c0 * cz;
dx = ptx00b - ex;
dy = pty00b - ey;
dz = ptz00b - ez;
spdx = spx - dx;
spdy = spy - dy;
spdz = spz - dz;
rptx = ptx00b + ex + spdx;
rpty = pty00b + ey + spdy;
rptz = ptz00b + ez + spdz;
rflx = rptx - ptx00b;
rfly = rpty - pty00b;
rflz = rptz - ptz00b;
lenrfl = sqrt(rflx * rflx + rfly * rfly + rflz * rflz);
vpdotrfl = (vpax * rflx + vpay * rfly + vpaz * rflz) / (lenvpa * lenrfl);
theta = acos(vpdotrfl);
d0 = lenvpa * sin(theta);
if (systemNum == 850)
{
      shiftd = mgrad * (d0 - min_d_850);
}
if (systemNum == 851)
{
      shiftd = mgrad * (d0 - min_d_851);
}
if (systemNum == 852)
{
      shiftd = mgrad * (d0 - min_d_852);
}
if (systemNum == 853)
{
      shiftd = mgrad * (d0 - min_d_853);
}
```

```
                        blueD = 55.0;
                        greenD = 255.0;
                        redD = 55.0;
                        blueD += shiftd;
                        greenD += shiftd;
                        redD += shiftd;
                        btBlueInt = int(blueD);
                        btGreenInt = int(greenD);
                        btRedInt = int(redD);
                        countx++;
                        if (btBlueInt > 255)
                        {
                            btBlueInt = 255;
                        }
                        if (btBlueInt < 0)
                        {
                            btBlueInt = 0;
                        }
                        if (btGreenInt > 255)
                        {
                            btGreenInt = 255;
                        }
                        if (btGreenInt < 0)
                        {
                            btGreenInt = 0;
                        }
                        if (btRedInt > 255)
                        {
                            btRedInt = 255;
                        }
                        if (btRedInt < 0)
                        {
                            btRedInt = 0;
                        }
                        btBlue = System::Byte(btBlueInt);
                        btGreen = System::Byte(btGreenInt);
                        btRed = System::Byte(btRedInt);
                        countx++;
                        RedVp[scrnindx] = btRed;
                        GreenVp[scrnindx] = btGreen;
                        BlueVp[scrnindx] = btBlue;
                    }
                }
            }
            thetax += dphi_c;
        }
        phix += dphi_c;
    }
    //
    int stophere = 0;
}
//
void Objects_Cls::NextCoordinatesType3(double &ptx33P, double &pty33P, double &ptz33P, double &scrnx33P,
double &scrny33P, double ptx33p, double pty33p, double ptz33p)
```

```
{
    ptm03 = ptx33p;
    ptn03 = pty33p;
    pto03 = ptz33p;
    ptd03 = ptm03;
    pte03 = ptn03 * ne03 + pto03 * oe03;
    ptf03 = ptn03 * nf03 + pto03 * of03;
    pti03 = ptd03 * di03 + ptf03 * fi03;
    ptj03 = pte03;
    ptk03 = ptd03 * dk03 + ptf03 * fk03;
    ptx03 = pti03 * ix03 + ptj03 * jx03;
    pty03 = pti03 * iy03 + ptj03 * jy03;
    ptz03 = ptk03;
    ptm02 = ptx03 + T03x;
    ptn02 = pty03 + T03y;
    pto02 = ptz03 + T03z;
    ptd02 = ptm02;
    pte02 = ptn02 * ne02 + pto02 * oe02;
    ptf02 = ptn02 * nf02 + pto02 * of02;
    pti02 = ptd02 * di02 + ptf02 * fi02;
    ptj02 = pte02;
    ptk02 = ptd02 * dk02 + ptf02 * fk02;
    ptx02 = pti02 * ix02 + ptj02 * jx02;
    pty02 = pti02 * iy02 + ptj02 * jy02;
    ptz02 = ptk02;
    ptm01 = ptx02 + T02x;
    ptn01 = pty02 + T02y;
    pto01 = ptz02 + T02z;
    ptd01 = ptm01;
    pte01 = ptn01 * ne01 + pto01 * oe01;
    ptf01 = ptn01 * nf01 + pto01 * of01;
    pti01 = ptd01 * di01 + ptf01 * fi01;
    ptj01 = pte01;
    ptk01 = ptd01 * dk01 + ptf01 * fk01;
    ptx01 = pti01 * ix01 + ptj01 * jx01;
    pty01 = pti01 * iy01 + ptj01 * jy01;
    ptz01 = ptk01;
    ptm00 = ptx01 + T01x;
    ptn00 = pty01 + T01y;
    pto00 = ptz01 + T01z;
    ptd00 = ptm00;
    pte00 = ptn00 * ne00 + pto00 * oe00;
    ptf00 = ptn00 * nf00 + pto00 * of00;
    pti00 = ptd00 * di00 + ptf00 * fi00;
    ptj00 = pte00;
    ptk00 = ptd00 * dk00 + ptf00 * fk00;
    ptx00 = pti00 * ix00 + ptj00 * jx00;
    pty00 = pti00 * iy00 + ptj00 * jy00;
    ptz00 = ptk00;
    ptx00 += Tcgx;
    pty00 += Tcgy;
    ptz00 += Tcgz;
    si = ptx00 - vpx;
    sj = pty00 - vpy;
```

```
	sk = ptz00 - vpz;
	i = abs(si);
	j = abs(sj);
	k = abs(sk);
	s0i = ptx00;
	s0j = pty00;
	s0k = ptz00;
	if (i > 0.000000001)
	{
		mji = sj / si;
		mki = sk / si;
		tempi = (mki * s0i - s0k) / mki;
		tempj = mji * (tempi - s0i) + s0j;
	}
	else if (j > 0.000000001)
	{
		mij = si / sj;
		mkj = sk / sj;
		tempj = (mkj * s0j - s0k) / mkj;
		tempi = mij * (tempj - s0j) + s0i;
	}
	else if (k > 0.000000001)
	{
		mik = si / sk;
		mjk = sj / sk;
		tempi = mik * (-s0k) + s0i;
		tempj = mjk * (-s0k) + s0j;
	}
	ptx33P = ptx00;
	pty33P = pty00;
	ptz33P = ptz00;
	scrnx33P = tempi;
	scrny33P = tempj;
}
//
void Objects_Cls::IntersectScreen(double &scrnxP, double &scrnyP, double ptxP, double ptyP, double ptzP)
{
	si = ptxP - vpx;
	sj = ptyP - vpy;
	sk = ptzP - vpz;
	i = abs(si);
	j = abs(sj);
	k = abs(sk);
	s0i = ptxP;
	s0j = ptyP;
	s0k = ptzP;
	if (i > 0.000000001)
	{
		mji = sj / si;
		mki = sk / si;
		tempi = (mki * s0i - s0k) / mki;
		tempj = mji * (tempi - s0i) + s0j;
	}
	else if (j > 0.000000001)
```

```
    {
        mij = si / sj;
        mkj = sk / sj;
        tempj = (mkj * s0j - s0k) / mkj;
        tempi = mij * (tempj - s0j) + s0i;
    }
    else if (k > 0.000000001)
    {
        mik = si / sk;
        mjk = sj / sk;
        tempi = mik * (-s0k) + s0i;
        tempj = mjk * (-s0k) + s0j;
    }
    scrnxP = tempi;
    scrnyP = tempj;
}
//
```

EXHIBIT 104

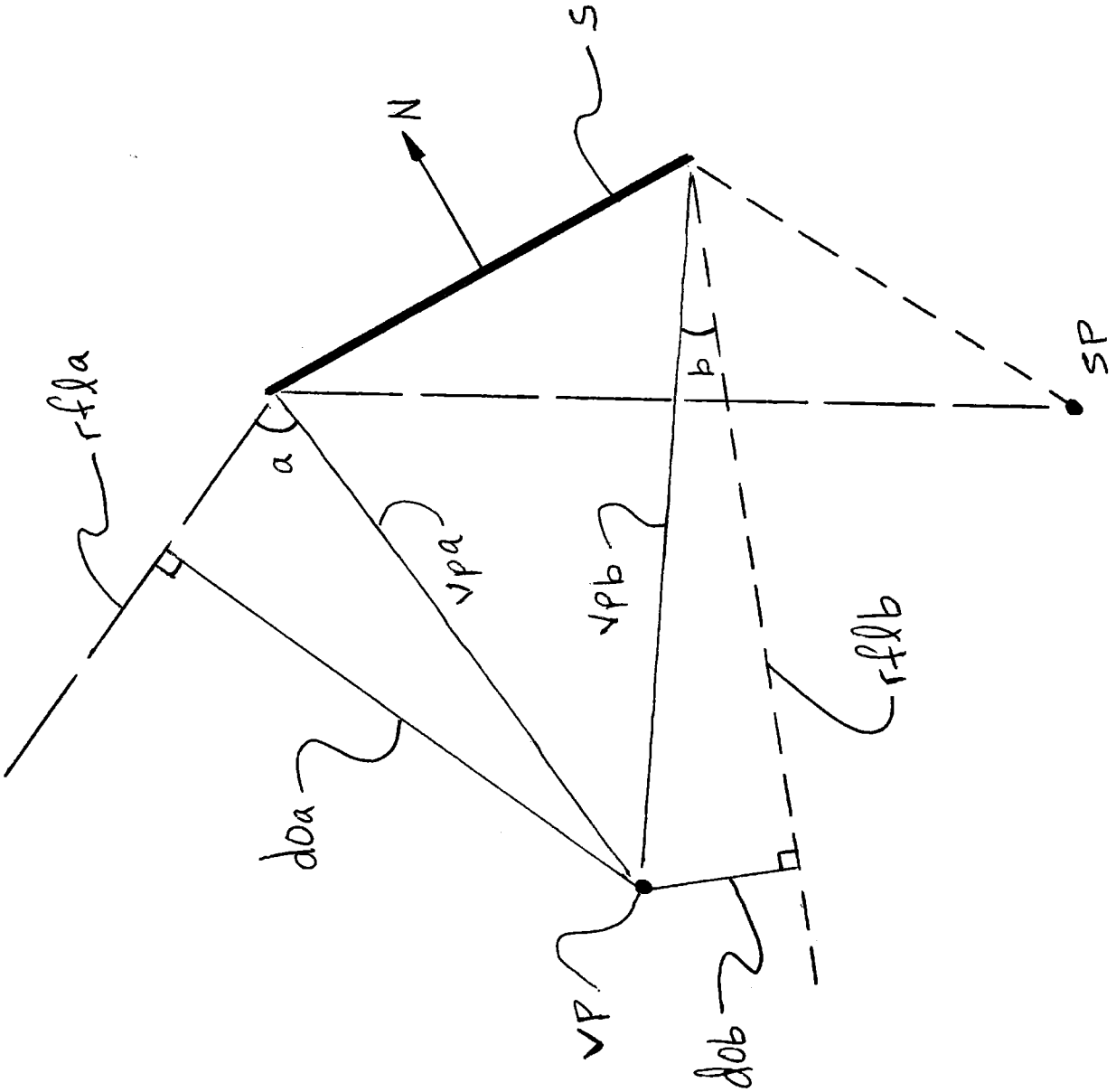
```
//
//
//
//
0000    rflx = rptx - ptx00a;
0001    rfly = rpty - pty00a;
0002    rflz = rptz - ptz00a;
0003    lenrfl = sqrt(rflx * rflx + rfly * rfly + rflz * rflz);
0004    vpdotrfl = (vpax * rflx + vpay * rfly + vpaz * rflz) / (lenvpa * lenrfl);
0005    theta = acos(vpdotrfl);
0006    mgrad =  -50 / (max_d - min_d);
0007    d0 = lenvpa * sin(theta);
0008    shiftd = mgrad * (d0 - min_d);
0009    blueD = 100.0;
0010    greenD = 255.0;
0011    redD = 100.0;
0012    blueD += shiftd;
0013    greenD += shiftd;
0014    redD += shiftd;
//
//
//
//
// Copyright 2017 by Louis A. Coffelt, Jr.
// TITLE OF THIS WORK: Photorealistic Surface Shading by Reflective Intensity 2017
// TYPE OF WORK: Computer Program
//
// This work is based on the Work by Louis A. Coffelt, Jr. created on
// Wednesday, October 20, 2010, 8:01:16 AM titled:
// ("Realistic 3D Surface Shading by Reflective Intensity 2010 ")
// Application Date: 5/13/2017.
// Service Request #: 1-5121154211
//
// This work is used in a larger work titled ("CAD Reflective Intensity")
// Application Date: December 13, 2016
// Service Request #: 1-4249380951
//
// The Claimed Work is the c++ program above at lines 0000 through 0014
// A description of this Claimed Work is the following:
//
//
//
//
// An objective of this claimed Work includes to derive photorealistic 3D surface
// shading for any type surface. A 3D graphic object is identified by a
// mathematical equation. There is a View Point in the 3D scene. There is a point
// light source in the 3D scene. Light source Incident Vectors intersect the
// graphic object. Light is reflected from the graphic object (Reflected Vector).
// The angle of incidence is equal to the reflected angle. The Reflected Vector and
// View Point are used to derive the light intensity for each corresponding point on
// the graphic object.
//
//
// lines 0000 through 0002 are the x, y, z, components of the Reflected Vector rfl.
// line 0003, lenrfl is the length of the Reflected Vector rfl.
// line 0004, vpa is the vector between the View Point and the graphic object point.
// line 0004, vpdotrfl is the vector dot product of vpa and rfl (cosine of angle).
// line 0005, theta is the angle between vectors vpa and rfl.
// line 0006, max_d is the maximum distance between the View Point and vector rfl.
// line 0006, min_d is the minimum distance between the View Point and vector rfl.
// line 0006, -50 is a selected constant for maximum shift of the base surface color.
// line 0006, mgrad is slope of a linear equation, which derives the color shift value.
// line 0007, d0 is current distance between the View Point and vector rfl.
// line 0008, shiftd is the value of the color shift from the base color value.
// lines 0009 through 0011, the base color of the graphic object surface is assigned.
// lines 0012 through 0014, the base color is shifted in order to create the
//                      photorealistic surface shading gradient.
```

EXHIBIT 105

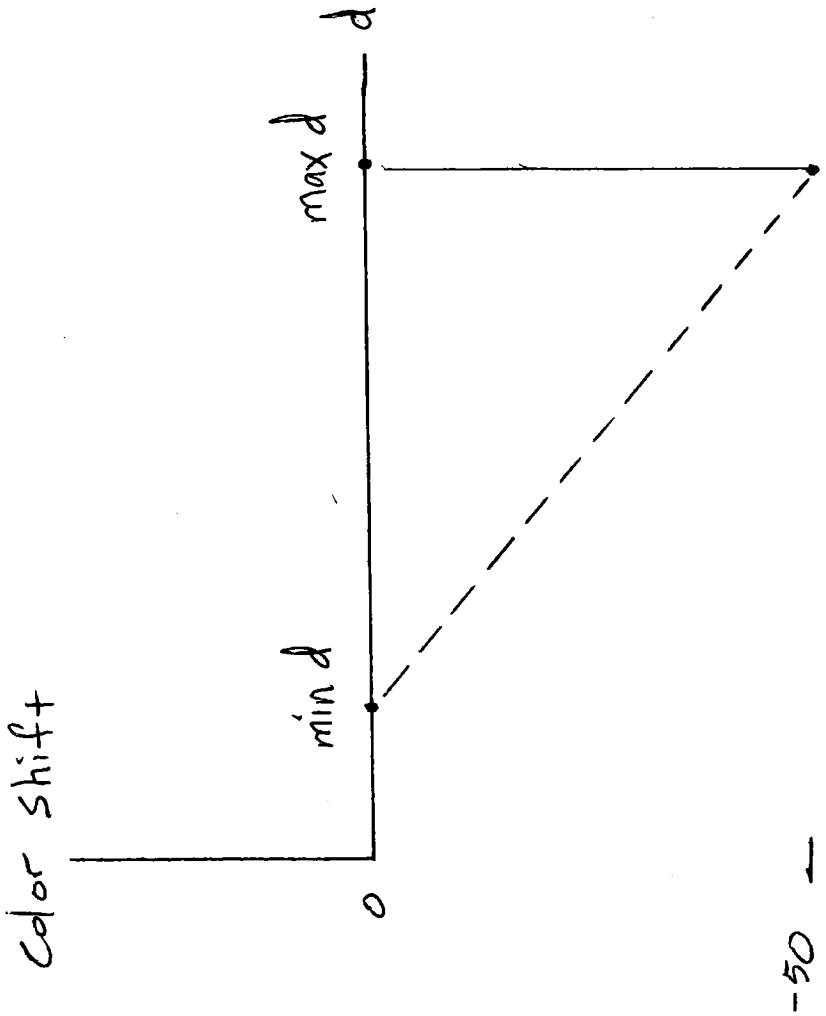
```
btGreenInt = int(btGreen);
btBlueInt = int(btBlue);

double redD_i = double(btRedInt);
double greenD_i = double(btGreenInt);
double blueD_i = double(btBlueInt);

double mgrad = -60.0 / (max_d - min_d);
double shiftd;

while (pty02a < plane_height)
{
    ptx02a = 0.0;
    while (ptx02a < plane_width)
    {
        NextCoordinatesType3(ptx00a, pty00a, ptz00a, scrnxA, scrnyA, ptx02a, pty02a, ptz02a);
        if (scrnxA > 0.0 && scrnxA < scrnWinches && scrnyA > 0.0 && scrnyA < scrnHinches)
        {
            scrncolxD = scrnxA * scrnppiD;
            scrnrowxD = scrnyA * scrnppiD;
            scrncolx = int(scrncolxD);
            scrnrowx = int(scrnrowxD);
            scrnindx = scrnrowx * scrnWpx + scrncolx;
            if (scrnindx < SizeVISv)
            {
                pVisDistA = VisibleV00[scrnindx];
                cVisDistA = sqrt(si * si + sj * sj + sk * sk);
                deltaVis = abs(cVisDistA - pVisDistA);
                if (deltaVis < 0.001)
                {
                    spax = spx - ptx00a;
                    spay = spy - pty00a;
                    spaz = spz - ptz00a;
                    lenspa = sqrt(spax * spax + spay * spay + spaz * spaz);
                    vpax = vpx - ptx00a;
                    vpay = vpy - pty00a;
                    vpaz = vpz - ptz00a;
                    lenvpa = sqrt(vpax * vpax + vpay * vpay + vpaz * vpaz);
                    bx = N00y * spaz - spay * N00z;
                    by = -(N00x * spaz - spax * N00z);
                    bz = N00x * spay - spax * N00y;
                    cx = N00y * bz - by * N00z;
                    cy = -(N00x * bz - bx * N00z);
                    cz = N00x * by - bx * N00y;
                    lenc = sqrt(cx * cx + cy * cy + cz * cz);
                    spadotc = (spax * cx + spay * cy + spaz * cz) / (lenspa * lenc);
                    phi = acos(spadotc);
                    spacos = lenspa * abs(cos(phi));
                    spasin = lenspa * sin(phi);
                    c0 = spacos / lenc;
                    ex = c0 * cx;
                    ey = c0 * cy;
                    ez = c0 * cz;
                    dx = ptx00a - ex;
```

```
dy = pty00a - ey;
dz = ptz00a - ez;
spdx = spx - dx;
spdy = spy - dy;
spdz = spz - dz;
rptx = ptx00a + ex + spdx;
rpty = pty00a + ey + spdy;
rptz = ptz00a + ez + spdz;
rflx = rptx - ptx00a;
rfly = rpty - pty00a;
rflz = rptz - ptz00a;
lenrfl = sqrt(rflx * rflx + rfly * rfly + rflz * rflz);
vpdotrfl = (vpax * rflx + vpay * rfly + vpaz * rflz) / (lenvpa * lenrfl);
theta = acos(vpdotrfl);
d0 = lenvpa * sin(theta);

blueD = blueD_i;
greenD = greenD_i;
redD = redD_i;

shiftd = mgrad * (d0 - min_d);

blueD += shiftd;
greenD += shiftd;
redD += shiftd;

btBlueInt = int(blueD);
btGreenInt = int(greenD);
btRedInt = int(redD);

if (btBlueInt > 255)
{
    btBlueInt = 255;
}
if (btBlueInt < 0)
{
    btBlueInt = 0;
}
if (btGreenInt > 255)
{
    btGreenInt = 255;
}
if (btGreenInt < 0)
{
    btGreenInt = 0;
}
if (btRedInt > 255)
{
    btRedInt = 255;
}
if (btRedInt < 0)
{
    btRedInt = 0;
}
```

EXHIBIT 106



COFFELT GRADIENT WORK



COFFELT GRADIENT WORK

EXHIBIT 107